Applications of Python in MSPA 400

Python Extensions

Python as a core language is at the center of thousands of applications. The language has been extended substantially into different areas. You can get an idea of what has evolved if you click on the Package Manager appearing on the Welcome Screen and view the list of installed and available packages. In MSPA 400, we will use three primary extensions: NumPy, Matplotlib and SciPy. Each will be discussed below.

NumPy

NumPy is an array-based approach to programming. It is designed for efficient computing. It is useful for data manipulation, analysis, computation and visualization. Many of the Python modules supplied as part of MSPA 400 use NumPy. The modules have many comments to indicate how the language is being used. For more in depth documentation, the Doc Browser is available on the Welcome Screen. There are certain notational conventions which need to be pointed out. While the interpreter (iPython) has the full functionality of NumPy, since we are using scripts in the course, NumPy must be imported. Importing of NumPy can be done in different ways resulting in code that appears different, but which does the same thing. It is important to recognize which convention is being used.

Convention (a): You will see the following statement in many programs on the internet and in publications: *import numpy as np*. When this statement appears, all functions in NumPy used by the program must have the letters: *np* as a preface to whatever function is involved. For example, if we wanted to convert a list to an array, one convention is the following:

```
import numpy as np
x = [1,2,3]
y= np.array(x).
```

This notation indicates the array function is being called from NumPy and converts the list x to a numpy array y.

Convention (b): To avoid the typing involved in adding *np* to all functions, the following is an equivalent way to proceed:

```
import numpy
from numpy import array
x = [1,2,3]
y = array(x).
```

This is the convention used in the educational materials from Canopy, but as indicated it is not the only convention in practice.

Convention (c): If a long list of functions is being used, an abbreviation is possible. The following will also work. The asterisk * indicates that all functions within NumPy are available to the script.

```
import numpy
from numpy import *
x = [1,2,3]
y = array(x).
```

Matplotlib

Various plotting statements are demonstrated in the Python modules. Much more is shown in the Canopy Editor Doc Manager. As with NumPy, Matplotlib must be imported in script and similar conventions are in use for this purpose. The following conventions are equivalent:

Convention (a):

```
import matplotlib.pyplot as plt
plt.figure()
x = [1,2,3]
y = [4,5,6]
plt.plot(x,y)
plt.show()
```

Convention (b):

```
import matplotlib.pyplot
from matplotlib.pyplot import plot, figure, show
figure()
x = [1,2,3]
y = [4,5,6]
plot(x,y)
show()
```

Convention (c):

```
import matplotlib.pyplot
from matplotlib.pyplot import *
figure()
x = [1,2,3]
y = [4,5,6]
plot(x,y)
show()
```

One of the features of Matplotlib is the capability to use color and different symbols when plotting. The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

SciPy

SciPy is a Python extension which utilizes NumPy to perform scientific and numerical computations. It includes an extensive list of packages which can be reviewed by accessing the Canopy Doc Manager.

Our use of SciPy will be limited to demonstrating differentiation and integration of polynomials. The following statement will be used to access the necessary SciPy program, poly1d.

```
import numpy
from numpy import poly1d
```

Help

Using the command prompt it is possible to access the help feature. For example, entering the phrase: *array?* at the command prompt will bring up information about the function. This is true for any function. Another example is entering the phrase: *plot?* to obtain information.

The command prompt can also be used to answer other questions. For example, if you need to find out the length of a list x, enter len(x) and the length will be displayed. If you need to find out the type of variable you are using, enter type(x) and the type will be displayed. The shape of an array can also be found using shape(x). These are just a few of the possibilities.